**Lab 4 Proximity Sensor – ECE 5780**

Nate Sheffield

A02268057

Nathan Critchfield

A02283426

**Objective**

The purpose of this lab is to implement a program using FreeRTOS to write to and read from a proximity sensor using UART and incorporate it into our audio amplifier system.

**Procedure**

For this lab we started with our completed Lab 3. We began by adding the setup for an additional UART port. We added the USART 3 on pin PC4 and PC5. This UART port we connect to the ultrasonic sensor. We added additional functions to our code for reading and writing to and from the added UART port. We then added the additional queues and support code to transfer the data from the UART port to the rest of the code. The additional code provided the functionality to move the receive the additional ‘t’ and ‘p’ commands from the PC, read the temperature or distance from the sensor and report that reading back to the PC. This functionality was all implemented without affecting the speaker and LED functionality of the project.

**Results**

For our lab to produce the desired results we had to modify our existing Lab 3 code to be able to incorporate the proximity sensor and connect to it via UART. In order to control both the input from the serial terminal and the proximity sensor we had to implement a second UART. For this lab we were primarily focused on being able to write to the proximity sensor and read from it either a temperature or a proximity measurement. In order to implement this we modified our code so that whenever a “t” was sent through the serial terminal our USART2 interrupt handler passes the data to our proximity task that then writes 0x50 to the sensor and then receives back a temperature measurement. If a “p” was sent our proximity tasks writes 0x55 to the sensor and receives back a distance measurement using ultrasonic sound waves. After writing either 0x50 or 0x55 to the sensor our USART3 read tasks polls for 1 byte of data in a temperature reading and for 2 bytes if it is a distance reading. Then it sends the data to our USART2 write function to write the temperature or distance reading to the serial terminal. Additionally, we wrote our code to be capable of running the proximity sensor and the audio amplifier circuit in such a way that even if we are polling for data from the sensor it does not inhibit the sound being produced from the audio amplifier circuit.

While writing the code for this lab we had lots of issues with trying to write to and read from the proximity sensor correctly. In our lab, we were able to successfully initialize USART3 for the proximity sensor and then connect that UART to the proximity sensor pins and then take the output from the sensor and pass it to the USART2 write function to write to the serial terminal. However, when we tried to write then read to the sensor we received no data and our code would stall waiting for data from the sensor. After some adjusting we were able to fix our USART3 write function to write the correct value to request a temperature or distance measurement but we still weren’t receiving data from the sensor. After a while we found that we had our RX and TX wires between the STM32 and our sensor. Normally for UART, the RX pins connects to the TX pin and vice versa. However, for the proximity sensor the TX pin is directly connected to the other TX and same for RX. After we realized this we were able to get the data written to our sensor but we still were unable to receive data. While debugging our code we commented out the polling function and we were able to write data to the serial terminal but it only contained our default value of 0 which meant we were not getting any data from the sensor. After lots of trial and error we found that the way we were trying to read the bytes from the sensor was not functioning so we redid our USART3 read code. We set it up so that if a “t” was sent we poll for 1 byte of data then send it to our USART2 write function and if a ”p” was sent we poll for 1 byte then shift it over by 8 then we poll for the 2nd byte and then send it to USART2 write. After making these changes we were finally able to receive data. However, we found that when we requested a proximity measurment our code would stall. This happened because it would poll and receive the 1st byte but then it would get interrupted by the task scheduler and exit the function and when it returned it would get stuck polling for the 2nd byte but now that byte did not exist anymore so it would stall the code. We then realized that because our LED, button and proximity sensor tasks were all the same priority that our proximity task would get interrupted in the middle of polling for data. We then called the vTaskPrioritySet function to change the priority to be higher while in the proximity task and then reset the priority to its original value as we left the proximity sensor task. This changed then worked great and we were able to request and receive temperature and distance data correctly.

A group of electronic devices with wires

Description automatically generated

**Figure 1. Proximity Sensor and Audio Amplifier Circuit connected to STM32 Nucleo Board**

A screen shot of a computer

Description automatically generated

**Figure 2. Temperature and Proximity Outputs from Proximity Sensor on Serial Terminal**

**Conclusion**

In conclusion, we learned a lot from the issues of this lab. We learned that for the proximity sensor the TX pin connects to the TX pin and the same for the RX pin. We also learned more about the importance of keeping track of your task priorities and how if all tasks have the same priority, they can interrupt other tasks in the middle when you do not want them to interrupt. We also learned about the importance of matching up your data types and making sure bites do not get over written when saving data from the proximity sensor. Likewise, it is important to make sure you are writing the data out correctly to the terminal to display the data in the right format. Overall, this was a challenging lab for us but in the end, we were able to accomplish all the lab requirements successfully without inhibiting any of our functions we implemented previously.

**Appendix**

***Main.c code***

1. #include "FreeRTOS.h"

2. #include "stm32l476xx.h"

3. #include "system\_stm32l4xx.h"

4. #include "task.h"

5. #include "timers.h"

6. #include "stdint.h"

7. #include "queue.h"

8. #include "stdio.h"

9.

10. #include "init.h"

11.

12. int main(void) {

13. //Initialize System

14. SystemInit();

15. clock\_Config();

16. but\_led\_queue = xQueueCreate(2,sizeof(uint8\_t));

17. but\_tim\_queue = xQueueCreate(2,sizeof(uint8\_t));

18. uart2\_to\_uart3\_queue = xQueueCreate(4,sizeof(uint8\_t));

19. //uart3\_prox\_sensor\_queue = xQueueCreate(8,sizeof(uint8\_t));

20. gpio\_Config();

21. USART2\_config();

22. USART3\_config();

23. timer\_Config();

24. DAC\_Config();

25.

26. //Set the priority for the interrupts

27. NVIC\_SetPriority(USART2\_IRQn,0x07);

28. NVIC\_SetPriority(TIM4\_IRQn,0x07);

29.

30. //Task for LED

31. if(xTaskCreate(LED\_task, "LED", 32, NULL, 2, NULL) != pdPASS){

32. while(1);

33. }

34. //Task for Button

35. if(xTaskCreate(Button\_task, "Button", 32, NULL, 2, NULL) != pdPASS){

36. while(1);

37. }

38.

39. if(xTaskCreate(prox\_sensor\_task, "Proximity Sensor", 256, NULL, 2, &prox\_handle) != pdPASS){

40. while(1);

41. }

42.

43. //Start Task Scheduler

44. vTaskStartScheduler();

45. while(1);

46. }

47.

48. //Function to toggle led\_state

49. void LED\_task(void \*pvParameters){

50. static uint8\_t buffer[1];

51. buffer[0] = 0;

52. while(1){

53. if(uxQueueMessagesWaiting(but\_led\_queue) > 0){

54. if(xQueueReceive(but\_led\_queue,buffer,50)== pdTRUE){}

55. }

56. //If the LED is off turn it on

57. if(buffer[0] == 1){

58. GPIOA->BSRR |= GPIO\_BSRR\_BS5;

59. }

60. //If the LED is on turn it off

61. else if(buffer[0] == 0){

62. GPIOA->BSRR |= GPIO\_BSRR\_BR5;

63. }

64. }

65. }

66.

67. //Function to read in button state and led\_state

68. void Button\_task(void \*pvParameters){

69. static uint8\_t buffer[1];

70. buffer[0] = 0;

71. uint32\_t button\_in;

72. while(1){

73. //Read in the value of the button

74. button\_in = GPIOC->IDR;

75. button\_in &= GPIO\_IDR\_ID13\_Msk;

76.

77. //If the button is pressed toggle the LED

78. if(button\_in == 0){

79. while(button\_in == 0){

80. button\_in = GPIOC->IDR;

81. button\_in &= GPIO\_IDR\_ID13\_Msk;

82. }

83. if(buffer[0] == 0){

84. buffer[0] = 1;

85. //Send led\_state to queue for LED Task

86. xQueueSendToBack(but\_led\_queue,buffer,50);

87. //Send led\_state to queue for TIM4\_IRQHandler

88. xQueueSendToBack(but\_tim\_queue,buffer,50);

89. }

90. else {

91. buffer[0] = 0;

92. //Send led\_state to queue for LED Task

93. xQueueSendToBack(but\_led\_queue,buffer,50);

94. //Send led\_state to queue for TIM4\_IRQHandler

95. xQueueSendToBack(but\_tim\_queue,buffer,50);

96. }

97. }

98. }

99. }

100.

101. void TIM4\_IRQHandler(void){

102. static uint32\_t sine\_count = 0;

103. static uint8\_t buffer[1];

104.

105. const uint16\_t sineLookupTable[] = {

106. 305, 335, 365, 394, 422, 449, 474, 498, 521, 541, 559, 574, 587, 597, 604,

107. 609, 610, 609, 604, 597, 587, 574, 559, 541, 521, 498, 474, 449, 422, 394,

108. 365, 335, 305, 275, 245, 216, 188, 161, 136, 112, 89, 69, 51, 36, 23,

109. 13, 6, 1, 0, 1, 6, 13, 23, 36, 51, 69, 89, 112, 136, 161,

110. 188, 216, 245, 275};

111.

112. //if there is a message waiting in the queue from ISR

113. if(uxQueueMessagesWaitingFromISR(but\_tim\_queue) > 0){

114. xQueueReceiveFromISR(but\_tim\_queue,buffer,NULL);

115. }

116. //if the LED is on

117. if (buffer[0] == 1){

118. sine\_count++; //Increment to the next value in the table

119. if (sine\_count == 64){

120. sine\_count = 0;

121. }

122. }

123. //Assign DAC to Sine\_Wave Table Current Value

124. DAC->DHR12R1 = sineLookupTable[sine\_count] + 45;

125. TIM4->SR &= ~TIM\_SR\_UIF; //Clears Interrupt Flag

126. }

127.

128. void USART2\_IRQHandler(void){

129. uint8\_t uart\_buffer[1];

130. uart\_buffer[0] = (uint8\_t)(USART2->RDR); //Get serial data

131. change\_note(uart\_buffer[0]);

132. //If uart\_buffer = t or p send into the queue

133. if (uart\_buffer[0] == 't' || uart\_buffer[0] == 'p'){

134. xQueueSendToBackFromISR(uart2\_to\_uart3\_queue,uart\_buffer,NULL);

135. }

136. }

137.

138. void change\_note(uint8\_t uart\_buffer){

139. if(uart\_buffer == 'a'){

140. TIM4->ARR = 0xFFFF008E; //2 MHz/(142) = 14.080 kHz interrupt rate; 220 Hz sine wave

141. }

142. else if (uart\_buffer == 'b'){

143. TIM4->ARR = 0xFFFF007E; //126; 246.94 Hz

144. }

145. else if (uart\_buffer == 'c'){

146. TIM4->ARR = 0xFFFF0077; //119; 261.63 Hz

147. }

148. else if (uart\_buffer == 'd'){

149. TIM4->ARR = 0xFFFF006A; //106; 293.66 Hz

150. }

151. else if (uart\_buffer == 'e'){

152. TIM4->ARR = 0xFFFF005E; //94; 329.63 Hz

153. }

154. else if (uart\_buffer == 'f'){

155. TIM4->ARR = 0xFFFF0059; //89; 349.23 Hz

156. }

157. else if (uart\_buffer == 'g'){

158. TIM4->ARR = 0xFFFF004F; //79; 392.00 Hz

159. }

160. else if (uart\_buffer == 'h'){

161. TIM4->ARR = 0xFFFF0046; //71; 440 Hz (High A)

162. }

163. }

164.

165. void prox\_sensor\_task(void \*pvParameters){

166. uint8\_t uart\_buffer[1];

167. uint16\_t measurement;

168.

169. while(1){

170. if(uxQueueMessagesWaiting(uart2\_to\_uart3\_queue) > 0){

171. if(xQueueReceive(uart2\_to\_uart3\_queue,uart\_buffer,50)== pdTRUE){

172. vTaskPrioritySet(prox\_handle,3);

173. if (uart\_buffer[0] == 't'){

174. USART3\_write(0x50);

175. while (!(USART3->ISR & USART\_ISR\_RXNE));

176. measurement = USART3->RDR;

177. USART2\_write(measurement,uart\_buffer);

178. }

179. else if (uart\_buffer[0] == 'p'){

180. USART3\_write(0x55);

181. while (!(USART3->ISR & USART\_ISR\_RXNE));

182. measurement = USART3->RDR;

183. measurement \*= 256;

184. while (!(USART3->ISR & USART\_ISR\_RXNE));

185. measurement += USART3->RDR;

186. USART2\_write(measurement,uart\_buffer);

187. }

188. vTaskPrioritySet(prox\_handle,2);

189. }

190. }

191. }

192. }

193.

194. void USART3\_write(uint8\_t measure\_type){

195. //Send command to measure the temperature or proximity distance

196. while(!(USART3->ISR & USART\_ISR\_TXE)); //Wait until hardware sets TXE

197. USART3->TDR = measure\_type & 0xFF; //Writing to TDR clears TXE Flag

198.

199. //Wait until TC bit is set. TC is set by hardware and cleared by software

200. while (!(USART3->ISR & USART\_ISR\_TC)); //TC: Transmission complete flag

201.

202. //Writing 1 to the TCCF bit in ICR clears the TC bit in ICR

203. USART3->ICR |= USART\_ICR\_TCCF; //TCCF: Transmission complete clear flag

204. }

217.

218. void USART2\_write(uint16\_t measurement,uint8\_t uart\_buffer[1]){

219. int nBytes = 18;

220. char serial\_message[18] = {0};

221. float inches = 0;

222.

223. //Produce appropriate string for temp or proximity measurement received

224. if (uart\_buffer[0] == 't'){

225. sprintf(serial\_message,"%i deg F\n\r",measurement);

226. }

227. else if (uart\_buffer[0] == 'p'){

228. inches = (float)(measurement / 25.4);

229. sprintf(serial\_message,"%.2f inches\n\r",(double)inches);

230. }

231. //Send Serial message to USART2 to send to serial terminal

232. for (int i=0; i < nBytes; i++){

233. while(!(USART2->ISR & USART\_ISR\_TXE)); //Wait until hardware sets TXE

234. USART2->TDR = serial\_message[i] & 0xFF; //Writing to TDR clears TXE Flag

235. }

236. //Wait until TC bit is set. TC is set by hardware and cleared by software

237. while (!(USART2->ISR & USART\_ISR\_TC)); //TC: Transmission complete flag

238.

239. //Writing 1 to the TCCF bit in ICR clears the TC bit in ICR

240. USART2->ICR |= USART\_ICR\_TCCF; //TCCF: Transmission complete clear flag

241. }

***Init.c code***

1. #include "FreeRTOS.h"

2. #include "stm32l476xx.h"

3. #include "system\_stm32l4xx.h"

4. #include "task.h"

5. #include "timers.h"

6. #include "stdint.h"

7. #include "queue.h"

8.

9. #include "init.h"

10.

11. void clock\_Config(void){

12. //Change System Clock from MSI to HSI

13. RCC->CR |= RCC\_CR\_HSION; // enable HSI (internal 16 MHz clock)

14. while ((RCC->CR & RCC\_CR\_HSIRDY) == 0);

15. RCC->CFGR |= RCC\_CFGR\_SW\_HSI; // make HSI the system clock

16. SystemCoreClockUpdate();

17.

18. //Turn Clock on for GPIOs

19. RCC -> AHB2ENR |= RCC\_AHB2ENR\_GPIOAEN;

20. //RCC -> AHB2ENR |= RCC\_AHB2ENR\_GPIOBEN;

21. RCC -> AHB2ENR |= RCC\_AHB2ENR\_GPIOCEN;

22. }

23.

24. void gpio\_Config(void){

25. //Set PA5 to output mode for LED

26. GPIOA->MODER &= ~GPIO\_MODER\_MODE5\_1;

27. GPIOA->MODER |= GPIO\_MODER\_MODE5\_0;

28. //Turn LED on

29. GPIOA->BSRR |= GPIO\_BSRR\_BS5;

30. //Set PC13 to input mode for Button

31. GPIOC->MODER &= ~GPIO\_MODER\_MODE13; //0xf3ffffff

32. }

33.

34. void timer\_Config(void){

35. //Turn on Clock for TIM4

36. RCC -> APB1ENR1 |= RCC\_APB1ENR1\_TIM4EN;

37.

38. //Enable interrupts for TIM4

39. NVIC->ISER[0] |= 1 << 30;

40. NVIC\_EnableIRQ(TIM4\_IRQn);

41.

42. TIM4->CR1 &= ~TIM\_CR1\_CMS; // Edge-aligned mode

43. TIM4->CR1 &= ~TIM\_CR1\_DIR; // Up-counting

44.

45. TIM4->CR2 &= ~TIM\_CR2\_MMS; // Select master mode

46. TIM4->CR2 |= TIM\_CR2\_MMS\_2; // 100 = OC1REF as TRGO

47.

48. TIM4->DIER |= TIM\_DIER\_TIE; // Trigger interrupt enable

49. TIM4->DIER |= TIM\_DIER\_UIE; // Update interrupt enable

50.

51. TIM4->CCMR1 &= ~TIM\_CCMR1\_OC1M;

52. TIM4->CCMR1 |= (TIM\_CCMR1\_OC1M\_1 | TIM\_CCMR1\_OC1M\_2); // 0110 = PWM mode 1

53.

54. TIM4->PSC = 0x7; // 16 MHz / (7+1) = 2 MHz timer ticks

55. TIM4->ARR = 0xFFFF008E; // 2 MHz / (70+1) = 14.080 kHz interrupt rate; 64 entry look-up table = 220 Hz sine wave

56. TIM4->CCR1 = 0x23; // 50% duty cycle (35)

57. TIM4->CCER |= TIM\_CCER\_CC1E;

58.

59. //Enable Control Register 1 for Counting

60. TIM4->CR1 |= TIM\_CR1\_CEN;

61. }

62.

63. void DAC\_Config(void){

64. //Turn on Clock for DAC1

65. RCC -> APB1ENR1 |= RCC\_APB1ENR1\_DAC1EN;

66. //Configure DAC1 GPIO in Analog Mode 0x3

67. GPIOA->MODER |= GPIO\_MODER\_MODE4;

68. //Enable DAC1 Channel 1

69. DAC->CR |= DAC\_CR\_EN1;

70. }

71.

72. void USART2\_config(void){

73. //Enable PA2 (TX) and PA3 (RX) to alternate function mode

74. GPIOA->MODER &= ~(0xF << (2\*2));

75. GPIOA->MODER |= (0xA << (2\*2));

76. //Enable alternate function for USART2 for the GPIO pins

77. GPIOA->AFR[0] |= 0x77 << (4\*2); //set pin 2 and 3 to AF7

78. //High Speed mode

79. GPIOA->OSPEEDR |= 0xF<<(2\*2);

80. //Pull up mode for PA3 RX

81. GPIOA->PUPDR &= ~(0xF<<(2\*2));

82. GPIOA->PUPDR |= 0x5<<(2\*2); //Select pull-up

83. //GPIO Output type: 0 = push-pull

84. GPIOA->OTYPER &= ~(0x3<<2);

85.

86. //Enable clk for USART2

87. RCC->APB1ENR1 |= RCC\_APB1ENR1\_USART2EN;

88. //Select system clock for USART2

89. RCC->CCIPR &= ~RCC\_CCIPR\_USART2SEL\_0;

90. RCC->CCIPR |= RCC\_CCIPR\_USART2SEL\_1;

91.

92. //Disable USART2

93. USART2->CR1 &= ~USART\_CR1\_UE;

94. //set data length to 8 bits

95. USART2->CR1 &= ~USART\_CR1\_M;

96. //select 1 stop bit

97. USART2->CR2 &= ~USART\_CR2\_STOP;

98. //Set parity control as no parity

99. USART2->CR1 &= ~USART\_CR1\_PCE;

100. //Oversampling to 16

101. USART2->CR1 &= ~USART\_CR1\_OVER8;

102. //Set up Baud rate for USART to 9600 Baud

103. USART2->BRR = 0x683; //1D4C

104. //USART2 Enable Receiver and transmitter

105. USART2->CR1 |= (USART\_CR1\_TE | USART\_CR1\_RE);

106.

107. //Enable interrupt for USART2

108. NVIC\_EnableIRQ(USART2\_IRQn);

109. //Enables interrupts for USART RX

110. USART2->CR1 |= USART\_CR1\_RXNEIE;

111.

112. //Enable USART2

113. USART2->CR1 |= USART\_CR1\_UE;

114.

115. //Verify that USART2 is ready for transmission

116. while ((USART2->ISR & USART\_ISR\_TEACK) == 0);

117. //Verify that USART2 is ready for reception

118. while ((USART2->ISR & USART\_ISR\_REACK) == 0);

119. }

120.

121. void USART3\_config(void){

122. //Enable PC4 (TX) and PC5 (RX) to alternate function mode

123. GPIOC->MODER &= ~GPIO\_MODER\_MODE4\_0;

124. GPIOC->MODER |= GPIO\_MODER\_MODE4\_1;

125. GPIOC->MODER &= ~GPIO\_MODER\_MODE5\_0;

126. GPIOC->MODER |= GPIO\_MODER\_MODE5\_1;

127. //Enable alternate function for USART3 for the GPIO pins

128. GPIOC->AFR[0] |= GPIO\_AFRL\_AFSEL4; //set pin 4 to AF7

129. GPIOC->AFR[0] &= ~GPIO\_AFRL\_AFSEL4\_3;

130. GPIOC->AFR[0] |= GPIO\_AFRL\_AFSEL5; //set pin 5 to AF7

131. GPIOC->AFR[0] &= ~GPIO\_AFRL\_AFSEL5\_3;

132. //High Speed mode

133. GPIOC->OSPEEDR |= 0xF << (2\*4);

134. //Pull up mode for PC5 RX

135. GPIOC->PUPDR &= ~(0xF<<(2\*4));

136. GPIOC->PUPDR |= 0x5<<(2\*4); //Select pull-up

137.

138. //Enable clk for USART3

139. RCC->APB1ENR1 |= RCC\_APB1ENR1\_USART3EN;

140. //Select system clock for USART3

141. RCC->CCIPR &= ~RCC\_CCIPR\_USART3SEL\_0;

142. RCC->CCIPR |= RCC\_CCIPR\_USART3SEL\_1;

143.

144. //Disable USART3

145. USART3->CR1 &= ~USART\_CR1\_UE;

146. //set data length to 8 bits

147. USART3->CR1 &= ~USART\_CR1\_M;

148. //select 1 stop bit

149. USART3->CR2 &= ~USART\_CR2\_STOP;

150. //Set parity control as no parity

151. USART3->CR1 &= ~USART\_CR1\_PCE;

152. //Oversampling to 16

153. USART3->CR1 &= ~USART\_CR1\_OVER8;

154. //Set up Baud rate for USART3 to 9600 Baud

155. USART3->BRR = 0x683;

156. //USART3 Enable Receiver and transmitter

157. USART3->CR1 |= (USART\_CR1\_TE | USART\_CR1\_RE);

158.

159. //Enable USART3

160. USART3->CR1 |= USART\_CR1\_UE;

161.

162. //Verify that USART3 is ready for transmission

163. while ((USART3->ISR & USART\_ISR\_TEACK) == 0);

164. //Verify that USART3 is ready for reception

165. while ((USART3->ISR & USART\_ISR\_REACK) == 0);

166. }